## REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>Unclassified | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION / AVAILABILITY OF REPORT<br>Approved for public release; distribution unlimited. |
| 2b. DECLASSIFICATION / DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>NRL Report 8829 | 5. MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a. NAME OF PERFORMING ORGANIZATION<br>Naval Research Laboratory<br>6b. OFFICE SYMBOL (*If applicable*) | 7a. NAME OF MONITORING ORGANIZATION |
| 6c. ADDRESS (*City, State, and ZIP Code*)<br>Washington, DC 20375-5000 | 7b. ADDRESS (*City, State, and ZIP Code*) |
| 8a. NAME OF FUNDING / SPONSORING ORGANIZATION<br>Naval Sea Systems Command<br>8b. OFFICE SYMBOL (*If applicable*) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
| 8c. ADDRESS (*City, State, and ZIP Code*)<br>Washington, DC 20362 | 10. SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO. 62712N; PROJECT NO. SF12131691; TASK NO.; WORK UNIT ACCESSION NO. DN 080-003 |

11. TITLE (*Include Security Classification*)
Architecture for a Demonstration Radar-Communication Link

12. PERSONAL AUTHOR(S)
J. O. Coleman

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (*Year, Month, Day*) | 15. PAGE COUNT |
|---|---|---|---|
| Interim | FROM 10/81 TO 4/84 | 1984 September 28 | 23 |

16. SUPPLEMENTARY NOTATION

| 17 COSATI CODES | | |
|---|---|---|
| FIELD | GROUP | SUB-GROUP |
| | | |
| | | |

18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
Data link control protocols; Computer communication networks; Data link control procedures; Tactical data transfer; Network architecture

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

For the best surveillance picture, surveillance data from sources on different platforms should be combined. This requires a robust communication capability for transferring both raw sensor data and processed track data between platforms. A communication system can be built using existing radar transmitters and antennas. This report describes a communications architecture and associated protocols used in a demonstration of such a system. The system outlined controls the interfacing of the communication and radar functions, controls system timing, and provides framing of messages and various levels of error control. The services provided by this architecture allow the communication link to be used by client systems with only minimal concern with the details of the link's operation. A brief description of possibly useful extensions to the system is included.

| | |
|---|---|
| 20. DISTRIBUTION / AVAILABILITY OF ABSTRACT<br>☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS | 21. ABSTRACT SECURITY CLASSIFICATION<br>Unclassified |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>C. E. Jedrey (SEA 6251) | 22b. TELEPHONE (*Include Area Code*)<br>(202) 692-9761<br>22c. OFFICE SYMBOL |

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

NRL Report 8829

# Architecture for a Demonstration Radar-Communication Link

JEFFREY O. COLEMAN

*Radar Analysis Branch*
*Radar Division*

September 28, 1984

NAVAL RESEARCH LABORATORY
Washington, D.C.

## CONTENTS

# ARCHITECTURE FOR A DEMONSTRATION RADAR-COMMUNICATION LINK

## INTRODUCTION

The Navy is pursuing the integration of surveillance data from sources on different platforms to provide an improved surveillance picture common to several platforms. For example, work is under way to combine data from radar and electronic-warfare support measures (ESM) equipment, possibly involving multiple platforms [1]. Success in these endeavors will require a robust communication capability for transferring both raw sensor data and processed track data between platforms. One way to communicate is to use the existing radar transmitters and antennas. Consequently, NRL investigated techniques for radar communications, and concluded in an initial study that such techniques are both practical and desirable [2]. The next step in the NRL program was to design and build a demonstration system. The basic goals of the demonstration radar-communication system were:

- to communicate using a surveillance radar without significantly degrading radar performance,
- to discover the types of capabilities that are important in such a system, both those capabilities involving only a single point-to-point link and those relating to a network of such links, and
- to provide a test bed for ideas relating to radar communication.

This demonstration system implements a communication link between NRL's Chesapeake Bay Detachment (CBD) and NRL's Tilghman Island Field Station located on Tilghman Island in the Chesapeake Bay. The transmitter of an experimental L-band surveillance radar at CBD is used along with a *radar emulator* at Tilghman Island. The communication receivers are separate from the radar. The basic approach is to replace several of the radar pulses with the output of a modem as the main beam of the radar's antenna sweeps over the receive site at Tilghman Island [2]. The emulator on Tilghman Island makes it appear to the communication receiver at CBD as though there were a radar transmitter on the island using the same technique to send data to CBD.

The generation of the material reported here was the first step in the design of those portions of the communication system lying outside the data path provided by the modems. The architecture described is an outline of techniques that can be used to control the interfacing with the radar, control system timing, and provide data framing and various levels of error control. Taken collectively, the services provided by the system components implemented according to this architecture allow the *user* or *application* software to use the communication link without concern with the details of the link's operation. Ideally, the link should appear as a transparent path for data transferred between software modules at different sites.

I emphasize the experimental nature of the architecture described in this report. The techniques described here are not necessarily the optimum ones for any particular application, however, they should be a suitable starting point for discussion, analysis, and experimentation from which workable systems may result.

Manuscript approved May 7, 1984.

## OVERVIEW

### Layered Approach

Modern computer communication networks are generally designed around layered architectures [3,4]. The services needed to provide a communication path between two points are provided by entities structured into layers. The applications, which run on the computers at two sites, communicate by using a communication path provided by the highest layer entity at each site. These two entities provide these services by using a predefined set of procedures called a *protocol* to communicate with each other through a more primitive communication path provided by the pair of entities of the next lower layer at the two sites. The entities of this lower layer in turn provide a path for the use of the higher layer entities by using another protocol to communicate with each other through a yet more primitive path provided by entities of an even lower layer. This layering continues until at the lowest layer two entities communicate with each other by using a physical channel. This can be summarized by stating that at each layer *peer* entities, entities at the same layer but at different sites, use a protocol to communicate through a path provided by peer entities of the next lower layer. Figure 1 illustrates this relationship among the various entities. The broken lines represent communication paths that are not physical channels but are really provided by the pair of entities immediately below. For example, in Fig. 1, the path between the two *High* entities is provided by the *Middle* entities. Therefore, the actual path of the data from the left *user* to the right *user* is down from the left *user* entity through the left *low* entity, across the physical channel to the right *low* entity, and up to the right *user*. A certain amount of overhead or control information is typically communicated between peer entities of the lower layers and is never seen by the *user* entities.

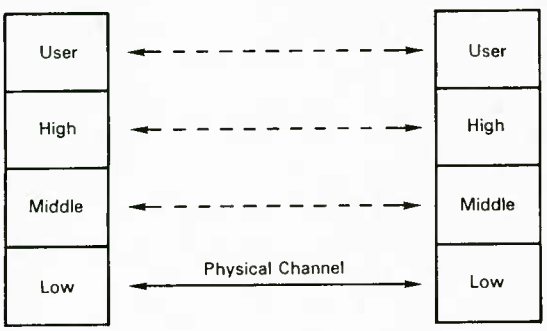


Fig. 1 — An example of layering

One interpretation of the structure in Fig. 1 is that entities at each successively higher layer add some capability, such as error correction, etc., to that provided by the entities at the layers below. The inverse viewpoint is that the entities at each layer exist to hide some information from the entities at higher layers. The information hidden might be, for example, restrictive timing requirements or a substandard reliability level of the underlying, lower-layer communication path. I prefer the information-hiding viewpoint for its more universal applicability. The information-hiding concept is generally regarded as an important tool in software design [5,6]. In the present context, the entities at each layer hide from entities at higher layers the internal details and methods of their own operation including the protocols they use for communicating with their counterparts at other sites. This allows many changes in the implementation of entities at one layer to proceed without the designer having to be concerned with effects on the operation of entities at other layers. The addition in capability provided by a layer can be viewed in information-hiding terms by recognizing that the entities at any one layer also hide from higher-layer entities the inadequacies of the services provided by the entities at lower layers.

### Terminology

In this report the term *layer* refers collectively to the cooperating peer entities at different sites. The terms *service* and *layer of service* are used interchangeably to refer to the complete set of capabilities

provided by interacting peer entities at different sites, usually a communication path of particular characteristics. Since each pair of interacting peer entities uses the services provided by the layer immediately below, the capabilities of a layer and hence service provided by that layer include in some sense the capabilities of all the lower layers. Entities that are implemented largely or entirely in software generally are referred to as *modules.* Although the specification of the interaction between peer entities is termed a protocol specification, as described earlier, the specification of the interaction between entities at adjacent layers at the same site is an *interface* specification. An *architecture* is a specific set of layers with associated service and protocol specifications. Interface specifications are generally considered part of an *implementation* of an architecture as the methods appropriate for interfacing modules are generally specific to a particular programming language, operating system, or even to specific computer hardware.

### Criteria for Layer Definition

Reference 7 lists a set of principles to consider in defining a set of architectural layers. A subset of those principles reflects the philosophy of this design.

- Create a boundary at a point where the service description can be small and the number of interactions across the boundary is minimized.
- Create separate layers to handle functions that are manifestly different in the process performed or the technology involved.
- Enable changes of functions or protocols within a layer without affecting the other layers.

Other things being equal, systems of many small (in the sense of functions performed) layers tend to have more communication overhead than systems of a few large layers because the many separate layers must operate independently. A system structured into a few large layers can result in efficient operation; however, the operation of such large layers is often difficult to understand because of complexity. In the early stages of development of this experimental system, simplicity and reliability are more important than efficiency. In addition, I have generally tried to have the functionally different aspects of the interaction with the radar transmitter occur from within different layers. These factors have resulted in more layers in the architecture described here than are used to accomplish roughly the same objectives in various well-known communication systems [4]. The entire system described here corresponds to roughly the combination of the data-link layer and the physical layer in one representative architecture [7]. Fortunately, most of the tasks to be divided between the layers were inherently sufficiently independent that little extraneous protocol overhead resulted from this approach. The many layers of this architecture should cause no serious implementation problems if the software is carefully designed to minimize the overhead involved in the interfaces between layers.

### Layers of the Architecture

This report discusses the layers of service whose functions are summarized in Table 1. Figure 2 shows the relationship among the various entities. The labels shown in Fig. 2 on the broken lines between peer entities refer to the designations I have given to the units of data that are passed through the associated communication paths. For example, the user-interface-modules at the top communicate through two separate paths provided by frame-service by passing back and forth units of data referred to as low-priority frames and high-priority frames. Where the units of data correspond in the physical operation of the system to some particular unit of radar transmission, the name was generally chosen to reflect this.*

*Viewed from a particular receiving site, once per *scan* the particular experimental radar we are using sends several *groups*, each of which contains several *pulses.*

Table 1 — Layers of the Architecture

| Layer | Function |
|---|---|
| User Interface | Provides multiplexing of multiple users (generally higher-level protocols) through the single link. Provides a convenient user interface. |
| Frame | Handles data frames of two priorities through a priority queue, detecting and discarding frames with transmission errors. |
| Codec | Does forward-error-correction to insure data integrity if a group is lost or garbled. |
| Scan | Handles units of data corresponding to radar scans. Deals with addressees and antenna direction. Detects missing groups. |
| Group | Handles units of data corresponding to radar pulse groups. Uses multiple pulses for each group to prevent collisions with radar output at the receive site. |
| Pulse | Handles units of data corresponding to radar pulses, removing random trailing data added by Modem. Provides data interface to hardware layers below. |
| Buffer | Handles bursts of data, transforming hardware data rates. Synchronizes outgoing bursts with the radar's timing requirements. |
| Modem | Handles high-speed bit streams, sending them through the RF channel, including the radar transmitter. |

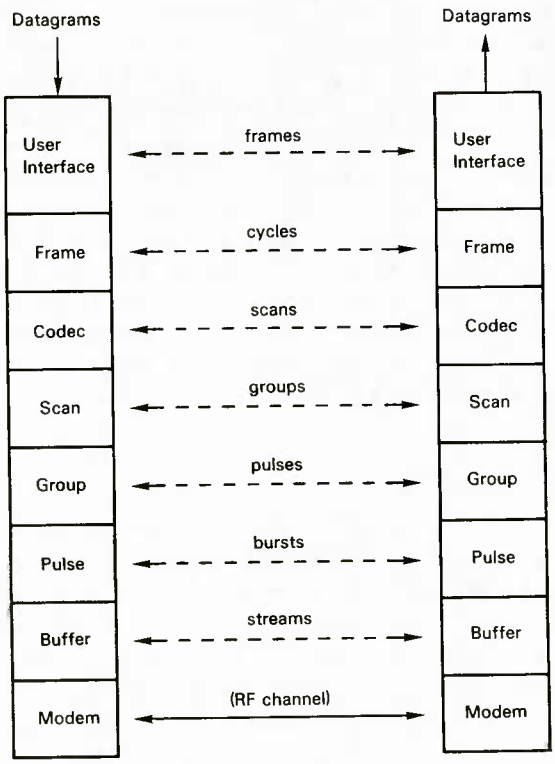


Fig. 2 — Relationships between the layers

Other layers, higher than those discussed here, would need to be defined in an operational network of these links.† For a simple demonstration link, however, these should be adequate. In addition, the layers detailed here will provide a reasonable basis upon which to build any higher layers that may be desired later.

Some of these layers are hardware, some software, and some include both. The modem and buffer layers must be implemented in hardware. The scan, group, and pulse layers should probably be largely implemented in software with hardware interfaces to the radar as well as to the buffer layer below. The user interface, frame, and codec layers can be entirely software with no hardware interfaces of any kind.

While this report explicitly discusses only a single link between two sites, there is no reason why this architecture cannot accommodate multiple two-site links using a single radar transmitter. The transmit-site entities can be (and should be) shared between links to several receive sites. A similar statement applies to a receive site that has links to several transmit sites. In addition to extending this link architecture for multiple-site operation, additional layers would need to be added above those presented here (or, possibly, between user-interface-service and frame-service), to make the system a usable network. The first such additional layer would almost certainly be a network layer. If this communication system were to be interoperable with other Department-of-Defense (DoD) computer-communication networks, a layer conforming to the DoD-standard *Internet Protocol* [8] specification should probably reside above the network layer. Further, the associated DoD standard *Transmission Control* [9] and *User Datagram* [10] protocols might be appropriate to provide virtual-circuit and datagram services.

The rest of this report discusses the layers of the architecture in detail, beginning with the lowest layer and proceeding to the highest. For each layer, I generally give the purpose of the layer's existence, a brief list of the resources available to the entities of that layer, and a suggested method or protocol by which the entities of that layer can provide the required services. As the scope of this report is limited to link architecture, the suggested methods given are not necessarily complete detailed implementation plans. For the layers that would likely have hardware implementations, I have presented only a general collection of ideas that may (or may not, depending on hardware constraints) prove useful to the hardware designers. Enough detail is provided for most of the layers, however, to show that an implementation providing the required services is possible, thus removing most of the technological risk.

## MODEM SERVICE

### Purpose

The modem provides a means of sending streams of high-speed data through the RF channel provided by the radar system. Because of implementation constraints, the modem can be expected to append some random bits to the end of the data burst.

### Available Resources

Since the modem is the lowest layer of the architecture, it has no lower layer of communication services upon which to draw. It is given only an RF port into the radar transmitter and a simple antenna and amplifier combination for a receiver.

### Suggested Method

The modem is a self-contained piece of hardware whose detailed theory of operation is beyond the scope of this report. The discussion below centers on the interface to buffer service. Although this

†Probably, as a minimum, equivalents of the *network* and *transport* layers in the ISO model [7] would be needed.

report generally omits discussion of interface issues, I am making an exception for hardware entities where discussing interfacing helps make feasibility more apparent.

The modem interface lines described in the following sections are a subset of the EIA RS-449 Category I circuits [11] and were chosen as the minimum necessary to achieve the desired modem capability.

*At the Transmit Site*

The transmit portion of the modem has a 3-line interface with the buffer: request-to-send (RS), send-timing (ST), and send-data (SD). The buffer provides a clock signal to the modem using the ST signal at all times. When the transmit site buffer has data it wishes to pass to the receive site buffer, the sequence of events is as follows:

- The buffer activates RS and begins placing the outgoing data on the SD line at the next clock pulse of the ST line.

- When the modem sees the RS line activate, it begins accepting data using the SD and ST lines. Rather than transmitting the data immediately, however, the modem *stores* the data while sending through the RF channel (including the radar) a preamble to prepare the receive-site modem. This preamble consists of an alternating prefix (1s and 0s) followed by a fixed sync word (e.g. as in Ref. 12).

- Once the preamble has been transmitted, the modem begins removing the data from storage and transmitting it across the channel at the rate given by SD. Because data are entering and leaving the modem at the same rate, the modem will always be maintaining in storage the most recent data accepted from the buffer.

- After the buffer has clocked the last bit of data to the modem, the buffer deactivates the RS line. On seeing the RS line deactivate, the modem stops accepting data from the SD line and simply transmits the data it already has in storage. Once the data have been transmitted, the modem may continue to provide an arbitrary RF output for the duration of the prescheduled radar pulse.

*At the Receive Site*

The receive site modem-to-buffer interface consists of 3 lines: receive timing (RT), receive data (RD), and receive ready (RR), all of which are provided by the modem. The buffer is assumed to be ready to receive data at all times. When the modem activates the RR line, it indicates to the buffer that the first bit of received data should be sampled from the RD line on the next transition of the RT signal, which serves as a clock. When the modem deactivates the RR signal, it indicates to the buffer that the last bit of data sent by the transmit-site buffer has already appeared on the RD line. It *does not* imply that the last bit of data sent by the transmit-site buffer was the last bit clocked into the receive-site buffer before the RR was deactivated. On the contrary, the receive-site modem is presumed to have appended an unknown number of random bits to the end of the data stream being passed between the buffers.*

## BUFFER SERVICE

### Purpose

Buffer service provides a hardware communication path that can be interfaced to higher layers at a data rate compatible with computer I/O devices (unlike the modem, which operates at much higher

*These random bits will eventually be removed by higher-layer entities at the receive site.

speeds). Enough buffering is provided at the receive site to allow for the receipt of more than one burst of data from the modem in rapid succession.

**Available Resources**

The only communication resource available to the buffer is the modem. The transmit-site buffer has four interfaces to the radar. They are:

- lines from the radar that provide antenna direction,
- lines from the radar indicating pulse ID within a radar pulse train,
- a line from the radar, *radar-pulse*, giving the starting time of a radar pulse, and
- a line to the radar, *radar-function*, controlling the RF switch that determines whether a particular radar pulse will be used for radar or communication.

The receive-site buffer has no interfaces with the radar.

**Suggested Method**

The buffer must be implemented in hardware. Ideally, care should be taken to ensure that the design is easily extendible to either faster modem data rates, larger numbers of bits per burst, or both.

*At the Transmit Site*

The transmit-site buffer accumulates one burst worth of data from the pulse-module in a high-speed hardware memory of length *buffer length.** The length of the burst is assumed to be less than or equal to *buffer length.* Associated with the data burst is a pulse ID and an azimuth before which it must not be transmitted. The buffer waits for the azimuth to arrive as shown by the lines from the radar. Once the actual azimuth matches that specified with the data burst, the buffer begins watching the pulse ID lines from the radar. When the correct pulse is indicated, the buffer uses *radar pulse* to trigger the change of *radar function* from radar to communication, and it then sends the data burst in the hardware memory through the modem to the receive-site buffer. When *radar pulse* is deactivated, the buffer must change *radar-function* back to radar.†

*At the Receive Site*

The receive-site buffer must queue bursts from the modem in a hardware memory‡ from which they can be transferred to the pulse-module in the computer. Recall that the division of the data from the modem into bursts is defined by the behavior of the RR line. The burst boundaries must be preserved. Since the modem receiver can add meaningless bits to the end of a burst, the lengths of the bursts received at the receive-site buffer could be considerably longer than the corresponding transmitted data bursts. It is worth noting, however, that the length of the useful portion of the burst will never exceed *buffer length.* Further details of buffer implementation are not discussed here.

---

*If a sufficiently high-speed DMA path is available from the computer in which the higher-layer modules are implemented, this buffer may be part of the computer's main memory.

† *buffer length* (presumably equal to the longest possible data burst to be transmitted) should be chosen such that the time taken to transmit *buffer length* bits is just slightly less (for a safety margin) than the longest radar pulse that will ever be used for communication.

‡Again, if a sufficiently high-speed DMA channel is available, this could be part of the computer's main memory.

## PULSE SERVICE

### Purpose

Pulse service hides from higher layers the details of the hardware interface to the buffer. It also hides the addition of meaningless bits to the end of a burst during its passage through buffer service.

### Available Resources

Pulse service has available only the services provided by buffer service and has no interfaces to the radar.

### Suggested Method

The details of the hardware interface to the buffer are implementation dependent and will not be discussed at this time. The problem of eliminating the meaningless bits added by buffer service is just one of marking, somehow, the meaningful portion of the burst. There are two basic approaches to this: (a) appending a length indication at the beginning of the burst, and (b) using a special marker in the data to mark the end. As the latter involves either taking a chance that the chosen marker will appear in the data by accident or using complicated procedures to reversibly alter the data to prevent such occurrences, the former alternative was chosen for simplicity. It is also easier to protect the length information against errors using the explicit length indication.

*At the Transmit Site*

The transmit-site pulse module takes an 11-bit indication of the length of the data pulse to be transmitted, codes it into the 15-bit Hamming (error-correcting) codeword [13], appends it to the beginning of the data, and then transmits the data with coded length attached through buffer service to the pulse module at the receive site.

*At the Receive Site*

On receipt of a burst, the receive-site pulse module should remove the first 15 bits, compute from them the 11-bit length indication using the Hamming decoding algorithm, and pass the indicated number of data bits up to the group-module.

## GROUP SERVICE

### Purpose

Group service can send a single group of data through pulse service while attempting to hide from higher layers the possibility of a collision at the receive site with an outgoing radar pulse.

### Available Resources

The only communication resource available to the group module is pulse service. The transmit-site group module has an input from the radar, *radar pulse*, described earlier with pulse service. In addition, the group module at the receive site has a single interface line to the radar, *radar delay*, by which it can force the radar to delay the transmission of its next pulse by a predetermined fixed amount. This line or its software equivalent is also monitored by the transmit portion of the receive-site group module.

The receive-site group module will need the time-of-arrival of each incoming data pulse. How this information is obtained is an implementation-dependent issue and will not be discussed further here.

### Suggested Method

*At the Transmit Site*

The transmit-site group module accepts a data group consisting of up to *buffer length*—15 bits of data from the scan module and transmits it at the earliest opportunity to the receive-site group module in the following way:

- It sends a *null* (zero data bits) *warning pulse* through pulse service to warn the group module at the receive site of the impending *actual* (data-containing) data pulse.
- It then sends the actual data pulse through pulse service. The time between the null warning pulse and the actual data pulse is predetermined, identical for all transmit sites, and known to the receive-site group modules in advance.

The radar used in the demonstration normally transmits a pulse group consisting of a particular pattern of short-range pulses followed by a long-range pulse. The group module should replace the *first* of the radar's short-range pulses with the warning pulse, and it should replace the long-range pulse with the actual pulse.*

The action to be taken when a *radar-delay* indication is received locally is described below.

*At the Receive Site*

On receipt of a null data pulse the receive-site group module performs a calculation to see whether the locally transmitting radar will interfere with the receipt of the associated actual data pulse. If interference is projected, the group module activates *radar delay* causing the radar to delay its next pulse, thus removing the problem.† If the spacing between the warning pulse and the actual data pulse is chosen properly relative to the radar pulse spacing, it will be impossible for the local radar to interfere with both the warning pulse and the actual data pulse.

When the receive portion of the group module activates *radar delay* it should also signal the transmit portion of the group module. If the transmit group module happens to be in the situation in which it has passed a null warning pulse to the pulse module but has not yet passed the actual data pulse, the incoming actual data pulse will be lost as the timing of the outgoing actual data pulse cannot be changed. The next radar group should be delayed however to prevent an immediate repeat of the situation.‡

## SCAN SERVICE

### Purpose

Scan service provides a method of sending a scan of data groups corresponding to a radar scan from one site to another while hiding from higher layers both the physical locations of the receive sites and the direction in which the radar antenna is pointed. The groups of data may contain from 1 to *buffer-length*—30 bits each. Either 8 or 15 groups are available per scan, depending on the radar scan rate.

*Ideally, the spacing between the two data pulses would not be exactly the same as the normal spacing between the corresponding radar pulses. See below.

†Of course the newly delayed next pulse of the radar may now pose a threat to some *other* group arriving from a different site. Dealing with conflicts of this sort is beyond the scope of the planned demonstration system, but such problems must be dealt with in a system intended for operational use.

‡It may also be desirable to modify the radar's scan rate slightly to prevent this problem from repeating on the next scan; however, this is beyond the scope of the planned demonstration.

**Available Resources**

The only communication resource available to scan service is group service. The transmit-site scan module has access to the radar's antenna direction. It also has the relative positions of all receive sites, provided from a source external to the communication system proper.

**Suggested Method**

*At the Transmit Site*

A short time before the radar antenna reaches the azimuth of any receive site, the scan module asks the codec module for a scan of data destined for that particular receive site. If the codec module is able to comply, scan service then prepares each group in the scan for transmission by appending several control fields to the beginning of the data: *Group number* gives the position of the individual group within the scan, the first group having the *group number* zero. The *more* flag is set to zero in the last group and one in the rest of the groups. Table 2 gives the number of bits used by each of these parameters.

Table 2 — Numbers of Bits in Scan-Layer Control Fields

| Field | Bits |
|---|---|
| To | 3 |
| From | 3 |
| Group-number | 4 |
| More | 1 |

Each group with control fields then has its first 11 bits replaced by a 15-bit Hamming error-correcting codeword capable of correcting a single error. The completed groups are then sent through group service to the receive site.

*At the Receive Site*

When the receive-site scan service receives a group from group service it immediately decodes the Hamming-encoded first 15 bits, replacing them with the 11 corrected bits. Looking at the *To* field, it then determines whether it is the intended recipient of the group. If the group was intended for another site, it is discarded.

Once it has been determined that the group is at the correct receive site, the group is appended to any previously existing scan from the same transmit site as determined from *From.* If no scan from the site given by *From* already exists, one is started with the new group. Finally, the *More* flag is examined. If *More* is one, no further action need be taken until either more groups arrive or a time-out shows that no more are likely to.

When the end of a scan is indicated by either a time-out or by a zero value for *More*, the *group number* fields of the groups in the scan are examined. If the *group number* fields begin with zero, are consecutive, and the *More* flag of the last group is zero, it can be assumed that all the groups are present. Each group can then be stripped of its control fields and the scan can be passed up to codec service. If one or more groups are missing, a "null group" is substituted in its place before passing the scan of stripped groups to codec service.* Since the end of a scan is known by the value of a *More* flag instead of by a field indicating the total number of groups expected, the occurrence of a time-out implies an unknown number of group deletions from the end of the scan. Codec service will have to interpret a null last group accordingly.

*The exact nature of a null group is unimportant to the architecture and can be determined during implementation of the system.

## CODEC SERVICE

### Purpose

Codec service exists to provide a low error-rate path for a stream of data bits being passed between a frame module at one site and a frame module at another site. In addition, the transmit-site frame module provides a flag, *hurry*, which, if true, tells the codec module to minimize delay even at the expense of throughput.

### Available Resources

The only communication resource used by codec service is scan service, which allows transmission of scans of data groups to another site on a single scan of the radar. The transmit-site codec module must know which of two possible radar scan rates is in use.

### Suggested Method

*At the Transmit Site*

When the scan module requests a scan of data for transmission to a particular receive site, the codec module first checks to see whether it has any scans ready for transmission to that site. If the codec module has no data, it creates some by requesting data from the frame module and converting it to one or more scans. The next available scan can then be sent to the receive site codec module through scan service.

When the codec module does not have data to satisfy a scan module request, the codec module begins a *cycle* by requesting data from the frame module. Codec service has two modes of operation for which cycles have different data capacities: seven-group mode and fifteen-group mode, with the names indicating the number of groups used to transmit a cycle of data. Fifteen-group mode has a higher average throughput than seven-group mode but a longer average delay. Fifteen-group mode is always used when the radar is in its slow-scan mode, sending the 15 groups in a single scan. Either mode may be used when the radar is in its fast-scan mode. If fifteen-group mode is used while the radar is in fast-scan mode, a burst of 7 groups will be sent one scan, followed by a burst of 8 groups the next scan. In seven-group mode all 7 groups are always sent in a single scan. When the radar is in fast-scan mode the cycle mode is chosen on a cycle-by-cycle basis as follows: The codec module first requests the number of data bits from the frame module equal to the maximum capacity of a cycle in seven-group mode. If the frame module returns less than the requested number of bits, implying that there is no more data available, seven-group mode is used for its lower delay. If frame service returns exactly as many bits as requested, the mode decision is made according to the *hurry* flag provided with the data by the frame module, with seven-group mode used if *hurry* is true. If *hurry* is false, implying that fifteen-group mode should be used, the codec module should request an additional amount of data from the frame module corresponding to the difference in data capacity between a seven-group mode cycle and a fifteen-group mode cycle. Once the mode to be used and the amount of data to be sent have been determined, several steps remain in the conversion of the cycle of data to one or more scans. In the description of the procedure that follows, the reader should refer to Table 3 for the mode dependent information:

- The data to be transmitted should be padded on the right with the fewest possible arbitrarily chosen filler bits necessary to make the total length of the combination of padded data and *trash-length* equal an integral multiple of the length of a data word (see Table 3).

- The parameter *trash-length* should be set to the number of filler bits added above and then appended to the beginning of the padded data.

Table 3 — Mode-Dependent Codec Parameter Information

| Parameter | Seven-group mode | Fifteen-group mode |
|---|---|---|
| Bits of Trash-length | 3 | 4 |
| Bits of a data-word | 4 | 11 |
| Bits of a Hamming codeword | 7 | 15 |
| Code-mode | 5 | 10 |

max data bits per cycle ($|x|$ is number of bits in $x$): $|\text{data-word}| \times$ (Buffer-length − 31) − $|\text{Trash-length}|$

- Break the resulting data stream up into data words and encode each into a Hamming error-correcting codeword [13]. (See Table 3).

- Set the 4-bit *code-mode* parameter as shown in Table 3, encode it into a 7-bit Hamming codeword, pad it on the right with filler data as necessary to make its length equal to that of the data codewords, and append the padded *code mode* to the beginning of the coded data.

- Interleave the data into groups by using the $m$th bit of codeword $n$ as the $n$-th bit of group $m$.

- If operating in Fifteen-group mode with the radar in fast-scan mode, break the 15 groups into a scan of 7 groups followed by a scan of 8 groups. (The information-bearing part of the codeword containing *code mode* should be left in the first of the two scans.) Otherwise, use the entire set of groups as a single scan.

*At the Receive Site*

The codec module at the receive site is passed a scan's worth of data groups from the scan module. If a group is lost in transmission, the scan module makes the codec module aware of this and the codec module uses an all-zero group in place of the missing one. If the groups are not all the same length, all but the longest are padded with zeros to equalize the lengths. The first bit from each of the first 7 groups is collected into a 7-bit Hamming codeword and decoded to find the 4-bit quantity *code mode*. The mode of transmission is determined as the mode corresponding to the legal *code mode* value that is nearest (in Hamming distance) to the decoded value.

If fifteen-group mode is to be used and only a seven-group scan has been received, the codec module waits for the eight-group scan to arrive from the scan module before continuing. A timeout should be used so that if an eight-group scan never arrives the cycle can be aborted. When all 15 groups are present the first bit of each of the last eight is discarded.

Once the proper number of groups are present for the mode given by *code mode* and the first bit of each group has been removed as described, the following operations take place:

- The data are assembled into codewords by using the $n$th bit of group $m$ as the $m$th bit of codeword $n$.

- Each codeword is decoded into a data word.

- The data words are catenated together to form padded data.

- The parameter *trash length* is removed from the beginning.

- The number of filler bits specified by *trash length* is discarded from the (right) end of the padded data, and the data are passed to the frame module.

## FRAME SERVICE

### Purpose

Frame service provides for communicating data frames of two separate priorities through codec service while hiding from higher layers the unpredictability of the amounts of data moved by codec service at any one time. In addition, frame service discards frames that suffer transmission errors, delivering only those frames passing an error-detection test.

### Available Resources

Frame service uses only those resources provided by codec service.

### Suggested Method

The most fundamental peculiarity of codec service with which frame service must deal is the unpredictability of the amounts of data that codec service will request at a time. The transmit-site frame module cannot control the size of the frames of data it obtains as units from the user-interface module. Since codec service does not necessarily move frames as units, frame service must provide its own *frame demarcation.* In addition, the transmit-site frame module must maintain a queue to hold those portions of available frames that are more than the immediate data requirements of codec service. If the designation "high priority" is to have any meaning, the queue must be a priority queue [14], so that high-priority data do not get queued behind and thus transmitted after low-priority data already in the queue. The use of two priorities in this fashion implies that a new high-priority frame could become available at a time when an existing low-priority frame was partially transmitted, with part of the low-priority frame remaining in the queue. This would result in a high-priority frame being transmitted in the midst of a low-priority frame. The frame demarcation provided by the transmit-site frame module must therefore be sufficient to make the inserted frame recognizable by the receive-site frame module. The interaction between frame demarcation and the priority queue will make it desirable that the queue be placed at the output of the transmit-site frame module.

The basic technique used to handle frame demarcation is to insert a unique marker [15] or *flag sequence* into the data to separate frames. To achieve data transparency, it is then necessary to take special measures to prevent the accidental occurrence of the flag sequence in the data itself from being interpreted as the end of the frame. Carlson [16] describes the procedure used to achieve transparency in the Advanced Data Communication Control Procedures [17] as follows:

- The flag sequence is a unique eight-bit pattern (a 0-bit followed by six 1-bits ending with a 0-bit) used a synchronize the receiver with the incoming frame...

- To achieve transparency, the unique flag sequence is prohibited from occurring anywhere in the... information... by having the transmitter and receiver perform the following action...

- Transmitter: insert a zero bit following five contiguous one bits anywhere before sending the closing flag sequence (bit stuffing).

- Receiver: delete the zero bit following five contiguous one bits following a zero bit anywhere before receiving the closing flag sequence (destuffing).

This stuffing scheme, while adequate for a single data stream is insufficient for our purposes because it does not provide a mechanism for handling our two data streams, low and high priority, which will be intermixed at the output of the priority queue and hence at the output of the receive-site code module. What we *can* do is use the stuffing scheme just described as the basic synchronization mechanism for the low-priority data stream, using a flag sequence to signal the end of each frame. Now consider the output of the receive-site codec module. Because the insertion of a high-priority frame into the queue put it in front of any existing low-priority data, and that low-priority data may be only part of a frame (if codec service has already moved part of the frame), a high-priority frame can appear inserted into the low-priority data stream at an unpredictable place.* We therefore need a way to detect the beginning of a high-priority frame. This can be done by marking the beginning of each high-priority frame with a special "high-priority flag sequence." The end of the frame can then be marked with the same end-of-frame flag sequence used for the low-priority data. Since the low-priority data must also be protected from accidental occurrences of this high-priority flag sequence, it will be necessary to protectively stuff the low-priority data to protect against *both* flag sequences. In addition, it is desirable that the stuffing operations on the low- and high-priority data be identical if the flag sequence at the end of a high-priority frame is ever missed by the receiver (possible only with errors in transmission). In that case the flag sequence at the end of the surrounding low-priority frame would be recognized and would allow processing to revert to low-priority mode. A marking and stuffing algorithm based on these ideas is detailed below.

A technique based on these ideas was recently developed to do two-priority frame demarcation [18]. A concise description is given here of how this algorithm operates (in enough detail for implementation).

*At the Transmit Site*

When the transmit-site codec module requests data, the frame-module first tries to fill the request from the high-priority data in the output queue. If there are insufficient high-priority bits in the queue, the frame-module requests a high-priority frame from the transmit-site user-interface-module, processing it as described below before adding it to the queue. This process is repeated until either there are enough bits in the queue to satisfy the request or there are no more high-priority frames available. If more bits are needed after a request for a high-priority frame has been refused by the user-interface module, the frame module begins requesting low-priority frames, again processing them in a manner to be described before inserting them into the queue. If a request for a low-priority frame is refused by the user-interface module, the codec module is simply given fewer bits than it requested (possibly no data at all in the extreme case).

A peculiarity of codec-service for the transmit-site frame module is that codec service is known to be sometimes capable of trading off data throughput and transmission delay. To take advantage of this, the transmit-site frame module will provide with each set of bits passed to codec-service a flag called *hurry*, which will be set true if (and only if) the data delivered to codec service are known to include the last bit of high-priority data currently available. This can be recognized only by the refusal of the user-interface module to supply a high-priority frame.

The procedure to prepare a newly obtained frame for insertion into the output queue is:

- Compute a cyclic redundancy check (CRC) sequence for the frame using the CCITT V.41 generator polynomial [16],

  $$x^{16} + x^{12} + x^5 + 1,$$

  and append the check sequence to the end of the frame.

*The reverse is not true; since a high-priority frame can always be inserted into the priority queue as a unit, and since the priority queue assured us that codec service will always move all available high-priority data before moving any low-priority data, a high-priority frame will never have other data appear in its midst.

- Append a comma to the end of the frame and pass the frame to the finite-state machine (FSM) stuffer of Table 4. The table is interpreted as follows: The states of the machine are listed on the left, and the possible inputs are listed across the top. For each combination of state and input, the table shows the machine's next state. When output is called for, it is shown following the next state and separated from it by a slash. The first state in the table is the starting state of the machine.

- If (and only if) the frame is a high-priority frame, the output of the stuffer of Table 4 should be preceded by the binary high-priority flag sequence: 1111110.

- The accumulated output of the stuffer should be inserted into the priority queue according to the priority of the frame being stuffed.

Table 4 — The Two-Priority Stuffer

| State | Input = '0' | Input = '1' | Input = ',' |
|---|---|---|---|
| AA | AA/0 | BB/1 | AA/011111010 |
| BB | AA/0 | CC/1 | AA/011111010 |
| CC | AA/0 | DD/1 | AA/011111010 |
| DD | AA/0 | EE/1 | AA/011111010 |
| EE | AA/0 | AA/100 | AA/011111010 |

*At the Receive Site*

The data received from codec-service should be passed to the two-priority destuffing machine of Table 5. The format of Table 5 is similar to that of Table 4 except that an additional column labeled *saved* has been added. Saved is an auxiliary variable and will always contain the name of a state; it is initialized to AA, the starting state (only once, *not* once per frame). For (state) transitions on which input to the machine is equal to one, the machine operates as described above for the FSM of Table 4. For transitions on which the input to the machine is zero, the machine first determines the next state (without actually changing state) and generates the required output, if any. It then sets the variable saved to the value given by the saved column of Table 5 and changes state. If there is no entry in the saved column, the value of saved is left unchanged.

There are two output buffers, *low-priority buffer* and *high-priority buffer*, and two auxiliary flags, *low-priority error* and *high-priority error*. Output is initially put into low-priority buffer. Both auxiliary flags are initially clear. The exclamation point output shown with certain transitions of Table 5 should be interpreted as: begin putting data into high-priority buffer instead of low-priority buffer. The "err" output shown with certain transitions should be interpreted as: set the appropriate auxiliary flag, low-priority error or high-priority error, according to the output buffer currently in use. The comma output shown in the table indicates the end of a frame requiring that the following actions be performed before continuing to process input data through the destuffer:

- Test the error flag, low-priority error or high-priority error, corresponding to the buffer currently in use. If it is set, clear it. If it is already clear, compute a CRC sequence on all but the last 16 bits of the output buffer currently in use using the same generator polynomial used at the transmit site. The resultant check sequence should be compared to the last 16 bits of the buffer. If (and only if) they match, the contents of the buffer excluding the last 16 bits should be passed up to the user-interface module as a frame of the priority corresponding to the buffer in use.

- The contents of the buffer in use should be discarded, making the buffer available for another frame. The output of the two-priority destuffer of Table 5 should be directed to

Table 5 — Two-Priority Destuffer

| State | Input = '0' | Saved | Input = '1' |
|---|---|---|---|
| AA | AB | | BA |
| AB | AB/0 | | BB |
| BB | AB/01 | | CB |
| CB | AB/011 | | DB |
| DB | AB/0111 | | EB |
| EB | AB/01111 | | FB |
| FB | AG | | GB |
| AG | AA/011111 | | BG |
| BG | Saved/, | AA | CG |
| BA | AB/1 | | CA |
| CA | AB/11 | | DA |
| DA | AB/111 | | EA |
| EA | AB/1111 | | FA |
| FA | AM/11111 | | GA |
| AM | AA | | BM |
| — | — | — | |
| GB | AA/! | AB | HB |
| HB | AA/! | BB | IB |
| IB | AA/! | CB | JB |
| JB | AA/! | DB | KB |
| KB | AA/! | EB | LB |
| LB | AA/! | FB | AA/err |
| CG | AB/err | | DG |
| DB | AB/err1 | | EG |
| EG | AB/err11 | | FG |
| FG | AK/err111 | | GG |
| GG | AA/! | AG | HG |
| HG | AA/! | BG | AA/err |
| GA | AA/! | AA | HA |
| HA | AA/! | BA | IA |
| IA | AA/! | CA | JA |
| JA | AA/! | DA | KA |
| KA | AA/! | EA | LA |
| LA | AA/! | FA | AA/1err |
| BM | AB/err | | CM |
| CM | AB/err1 | | DM |
| DM | AB/err11 | | EM |
| EM | AB/err111 | | FM |
| FM | AA/err1111 | | GM |
| GM | AA/! | AM | AA/err |

low-priority buffer for successive data. (This may or may not be a change from the buffer previously in use.)

## USER-INTERFACE SERVICE

### Purpose

User-interface service is the highest layer of communication service outlined in this report and exists to provide users with a convenient-to-use facility for communicating *datagrams* of data. A

datagram is simply an ordered collection of zero or more data bits to be transferred as a unit from transmit-site user to receive-site user. Delivery is on a "best-effort" basis, that is, it is not guaranteed.* Datagrams that are received can be presumed error-free.

Conveniences provided by user-interface service might include:

- a way to specify the antenna azimuths associated with various receive sites,
- a way to control implementation-dependent features such as error logging,
- multiplexing multiple users onto the single channel by appending user ID numbers to the datagrams,
- queuing of outgoing datagrams from the time they are presented by the user until requested by frame service,
- queuing of user-provided data buffers to be used for incoming datagrams,
- temporary queuing of incoming datagrams when no user-provided receive data buffers are available, and
- the ability to cancel outgoing datagrams by removing them from the associated queue and to cancel requests to receive incoming datagrams by removing receive buffers from the receive-buffer queue.

### Available Resources

The only communication resources available to user-interface service are those provided by frame service.

### Suggested Method

The details of user-interface service are strongly implementation dependent and will not be discussed further.

## SUMMARY AND CONCLUSIONS

NRL is building a system to demonstrate computer-to-computer communication using the powerful transmitter and large rotating antenna of a surveillance radar. This communication capability can be used with only a small degradation in radar performance. The demonstration system illustrates ideas upon which an operational radar-communication system for a Navy task force could be built.

This report has outlined the layered architecture of NRL's demonstration radar-communication link, including some suggested protocols. This design would allow the users of the link to treat it as a nearly transparent data path. While this report concentrated on the architecture of a single two-site link, the extension of the architecture to encompass the participation of a site in multiple links (to multiple sites) is straightforward. Network and transport protocols could be added to a set of such links to produce a usable communication network.

*A guaranteed-delivery datagram service would require an acknowledgment/retransmission system so that datagrams lost enroute could be replaced. Such a system is beyond the scope of this project.

## REFERENCES


1. G. V. Trunk and J. O. Coleman, "Radar-ESM Correlation Decision Procedure," NRL Report 8476, May 7, 1981.

2. B. H. Cantrell, J. O. Coleman, and G. B. Trunk, "Radar Communications," NRL Report 8515, Aug. 26, 1981.

3. A. S. Tanenbaum, "Network Protocols," *ACM Computing Surveys*, **13**(4), 453-489 (1981).

4. P. E. Green, Jr., "An Introduction to Network Architectures and Protocols," *IEEE Trans. Communications*, **COM-28**(4), 413-424 (1980).

5. S. D. Hester, D. L. Parnas, and D. F. Utter, "Using Documentation as a Software Design Medium," *Bell System Tech. J.*, **60**(8) 1941-1977, (1981).

6. B. W. Kernighan and P. J. Plauger, *The Elements of Programming Style* (McGraw-Hill, New York, 1978), p. 65.

7. H. Zimmermann, "OSI Reference Model—The ISO Model of Architecture for Open Systems Interconnection," *IEEE Trans. Communications*, **COM-28**(4), 425-432 (1980).

8. Jon Postel, "Internet Protocol, DARPA Internet Program Protocol Specification," RFC-791, Network Information Center, SRI International, Menlo Park, CA 94025, September 1981.

9. Jon Postel, "Transmission Control Protocol," RFC-793, Network Information Center, SRI International, Menlo Park, CA 94025, September 1981.

10. Jon Postel, "User Datagram Protocol," RFC-768, Network Information Center, SRI International, Menlo Park, CA 94025, August 20, 1980.

11. *General purpose 37-position and 9-position interface for data terminal equipment and data circuit-terminating equipment employing serial binary data interchange*, EIA Standard RS-449. November 1977.

12. J. O. Coleman, "Optimum Synchronization Codes to Follow an Alternating Mark/Space Prefix," NRL Report 8494, July 31, 1981.

13. R. W. Hamming *Coding and Information Theory* (Prentice-Hall, Englewood Cliffs, N.J., 1980), pp. 39-42.

14. D. E. Knuth, *Sorting and Searching, The Art of Computer Programming, Volume 3* (Addison-Wesley, Reading, Mass., 1973), Section 5.2.3.

15. R. A. Scholtz, "Frame Synchronization Techniques," *IEEE Trans. Commun.* **COM-28**(8), 1204-1212 (1980).

16. D. E. Carlson, "Bit-Oriented Data Link Control Procedures," *IEEE Trans. Commun.* **COM-28**(4), 55-467 (1980).

17. ANSI Standard X3.66-1979, *Advanced Data Communication Control Procedures*, American National Standards Institute, New York.

18. J. O. Coleman, "A Two-Priority Frame-Synchronization Algorithm," NRL Report 8661, Feb. 23, 1983.




GPO 910-203